CMR Pull Request Review Checklist

This list is intended for people conducting code reviews of pull requests as described in the Code Change Helper. When reviewing a pull request the reviewer should use the following checklist to verify that the code will not break clients or cause problems and that the code/tests are correct, complete, follow conventions/guidelines, and can be merged into the master branch without issue. Some of the items on the list are very specific, e.g., no TODOs, while others are more open ended (well structured tests). The list is intended to make the job of reviewing pull requests easier and more repeatable by identifying areas in pull requests that commonly need addressing.

When reviewing a pull request, start at the top of the list and consider the most important things that could lead to problems. These are outlined as a set of questions in the *Most Important* section. Check off the boxes when you are satisfied that each of the criteria is met. Then proceed to the more specific items below in the *General*, *Testing*, and *Documentation* sections. Check off each box that is satisfied by the pull request and *a dd comments to the pull request for those that are not*. If all the boxes are checked then approve the request. Otherwise, mark it as *needing work*, or, if there are critical errors, *decline* it.

Most Important (Primum non nocere - "First, do no harm")

- [ ] Could this change break clients?
- [ ] Do the changes handle data that may have been previously saved or indexed by an older version of the code?
- [ ] What is the operational impact of this change - are there any potential issues such as special deployment procedures, performance issues, etc.?
- [ ] Are there tests for all cases (including edge cases)?
- [ ] What could go wrong?

General

- [ ] Does the code do what it's supposed to do?
- [ ] Have they implemented all of the acceptance criteria?
- [ ] There are no overly-long or complicated functions that should to be broken up for readability.
- [ ] Are web API parameters validated?
- [ ] Are symbols used rather than “magic number” constants or string constants? (OK in tests, particularly for error messages or response codes).
- [ ] There is no repeated or copy-and-paste code / tests.
- [ ] There are no TODOs.
- [ ] There are no stray `comment` blocks, commented out code, `capture`/`reveals`, `proto-saves`, `printlns`, or unnecessary logging.
- [ ] All namespaces and non-trivial `defs`/`defns`/`defmacros` have docstrings.
- [ ] There are no dangling requires, i.e., requires that were added and not used or requires that are no longer necessary due to code removal.
- [ ] CMR coding conventions are followed (indention and 100 column line limit, etc. See the CMR Coding Style Guidelines).
- [ ] Idiomatic Clojure is used (see the Clojure Style Guide).

Testing

- [ ] IF THE ISSUE IS A BUG FIX - a test was added that reproduces the conditions that triggered the bug.
- [ ] The tests are well structured and follow current practices (`are`/`are2`, etc.).

Documentation

- [ ] Documentation (api_docs.md, README.md, CMR Data Partner User Guide, and CMR Client Partner User Guide) was added for any new features or old documentation updated for any changed features. The partner guides are on the wiki so they cannot be updated as part of a pull request.
- [ ] Code/curl examples or sample data have been updated as necessary.
- [ ]

- [ ] Has the CHANGELOG been updated?